Povezivanje PC računala i robotskog kontrolera pomoću programskog jezika Python

Grđan, Kevin

Sveučilište
Sjever

Završni rad br. 030/MEH/2024

Povezivanje PC računala i robotskog kontrolera pomoću programskog jezika Python

Kevin Grđan, 0035223440

Varaždin, rujan 2024. godine

Sveučilište
Sjever

Odjel za mehatroniku

Završni rad br. 030/MEH/2024

Povezivanje PC računala i robotskog kontrolera pomoću programskog jezika Python

Student

Kevin Grđan, 0035223440

Mentor

Zoran Busija, dipl.ing.stroj.

Varaždin, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za mehatroniku

STUDIJ: preddiplomski stručni studij Mehatronika

PRISTUPNIK: Kevin Grđan | JMBAG: 0035223440

DATUM: 30.08.2024. | KOLEGIJ: Robotika

NASLOV RADA: Povezivanje PC računala i robotskog kontrolera pomoću programskog jezika Python

NASLOV RADA NA ENGL. JEZIKU: Connecting a PC and a robot controller using the Python programming language

MENTOR: Zoran Busija, dipl. ing. stroj. | ZVANJE: predavač

ČLANOVI POVJERENSTVA:

1. Siniša Švoger, mag.ing.mech, predavač
2. prof. dr. sc. Ante Čikić
3. Zoran Busija, dipl.ing.stroj, predavač
4. Josip Srpak, dipl.ing.el, viši predavač
5.

Zadatak završnog rada

BROJ: 030/MEH/2024

OPIS:

U završnom radu potrebno je:
- oblikovati dijelove pomoću 3D CAD programa
- dijelove uvesti u RobotStudio te time stvoriti novo okruženje
- objasniti pojam "socket communication"
- napisati RAPID program za izvođenje pokreta robota i komunikaciju s PC računalom
- napisati program za PC računalo u programskom jeziku Python
- simulirati komunikaciju između PC računala i kontrolera robota

Ključne riječi: 3D oblikovanje, RobotStudio, Python, "socket communication"

ZADATAK URUČEN: 05.09.2024. | POTPIS MENTORA: Busija Zoran

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

Predgovor

Rad je prikaz optimizacije tehničkog procesa koristeći više programskih jezika. U radu se koristi jednostavan proces koji je poboljšan koristeći prednosti programskih jezika kako bi se postigla veća efikasnost. Kroz rad se opisuje čitav proces stvaranja okruženja, opis načina razmjene podataka između programskih jezika i opis programskog koda.

Želim se zahvaliti svojoj obitelji, mentoru i prijateljima na savjetima i podršci bez kojih ovaj rad ne bi bio moguć.

Kevin Grđan

Sažetak

Ovaj rad prikazuje na koji način korištenje više programskih jezika unutar jednog projekta automatizacije može poboljšati efikasnost. Ovaj projekt koristi RAPID kod za upravljanje gibanja industrijskog robota i Python kod koji obrađuje unos korisnika na PC računalu i potrebne podatke šalje RAPID aplikaciji na robot kontroler. Također se obraća pozornost na internetsku komunikaciju putem *socketa* koja se koristi u radu i omogućava razmjenu podataka između različitih programskih kodova, odnosno različitih aplikacija.

Ovaj rad opisuje je radnu okolinu i 3D modele. Zatim slijedi opis komunikacije i završava sa obrazloženjem programskog koda.

Ključne riječi: 3D oblikovanje, RobotStudio, Python, „Socket communication“

Abstract

This work shows how the usage of multiple programming languages in a single automation process can increase efficiency. This project uses RAPID code for industrial robot movement control and Python code that interprets user input and sends necessary data to RAPID application. Focus is also on communication with sockets which enables data transfer between codes, or more accurately between applications.

This work describes work environment and 3D models. After that is description of communication used and ends with explanation of code.

Keywords: 3D modeling, RobotStudio, Python, „Socket communication“

Popis korištenih kratica

mm	milimetar
IRC5	Industrial robot controller 5
TCP	Transmission controll protocol

Sadržaj

1. Uvod 1
2. Opis zadatka i radne okoline 3
   2.1. Robot i kontroler 4
   2.2. 3D modeli 5
3. Komunikacija 9
4. RobotStudio okolina i RAPID kod 11
   4.1. Okolina 11
   4.2. RAPID programski kod 13
5. Python kod 26
6. Zaključak 35
7. Literatura 36

1. Uvod

U sve više povezanom industrijskom svijetu sve je češći spoj raznih industrijskih kontrolera i raznih softvera napisanih u različitim razvojnim okruženjima, u raznim programskim jezicima. Viši programski jezici omogućuju veliku fleksibilnost i ponekad je moguće napraviti kompleksnije programe nego što to dopušta programski jezik nekog industrijskog kontrolera. Industrijski kontroleri imaju znatno uži fokus primjene, te je zbog toga za njih ponekad teže napisati kompleksnije programe.

RAPID programski jezik razvijen od strane ABB-a je snažan programski jezik s velikim brojem naredbi i funkcija koje omogućavaju sve od preciznog postavljanja robota u određenu poziciju, konfiguraciju njegovih osi tijekom gibanja, te logičkih i ostalih funkcija koje olakšavaju programiranje rada kontrolera, odnosno samog robota. Nedostatak RAPID-a i sličnih programskih jezika je vrlo usko područje primjene. Takvi su jezici optimizirani za određene namjene, te mogu imati poteškoća kod razvoja programskih rješenja koja su van njihove namjene, te je u takvim slučajevima poželjno koristiti programske jezike za tu specifičnu namjenu ili programske jezike opće, odnosno široke namjene za razvoj programskog rješenja. S obzirom na to da je RAPID prvenstveno namijenjen za programiranje putanje i rada robota za neke ga je namjene poželjno koristiti zajedno s nekim drugim programskim jezicima.

Za razliku od RAPID programskog jezika Python je programski jezik široke namjene. Pythonov fokus na čitljivost programskog koda pomogao je u njegovoj sve široj implementaciji u području računalnog programiranja. Osim čitljivosti koda i lakoći korištenja jedan od najvećih doprinosa Pythonovom uspjehu je golem broj biblioteka koje pružaju gotove funkcije za širok spektar potreba. Iako bi bio loš izbor za programiranje putanje i ponašanja robotskih sustava Pythonova mogućnost obrade velike količine podataka, strojnog učenja i pristupa gotovim bibliotekama za upravljanje čini ga savršenim i sposobnim alatom u raznim primjenama, pa čak i u nekim tehničkim primjenama poput automatizacije.

Korištenjem oba programska jezika moguće je iskoristiti prednosti i jednog i drugog. Python ima znatno veću fleksibilnost, odnosno može lakše obraditi razne podatke, te se u njemu mogu razviti napredni algoritmi upravljanja koji mogu uključivati i generiranje putanje, kalibracija, računanje dinamike cijelog sustava i slično. Iako je u ovome radu korišten Python za izvršavanje algoritma i drugi se programski jezici mogu koristiti uz RAPID. Programski jezici poput MATLAB-a također mogu obavljati zadatke za koje jezici poput RAPID-a nisu optimalni. MATLAB se sa svojim dodatnim programskim paketima može koristiti za vrlo kompleksne i vrlo zahtjevne matematičke obrade koje neki drugi programski jezici poput RAPID-a ne bi mogli uopće izvršiti ili bi implementacija toga bila vrlo zahtjevna, te nepotrebno usporila izvođenje koda.

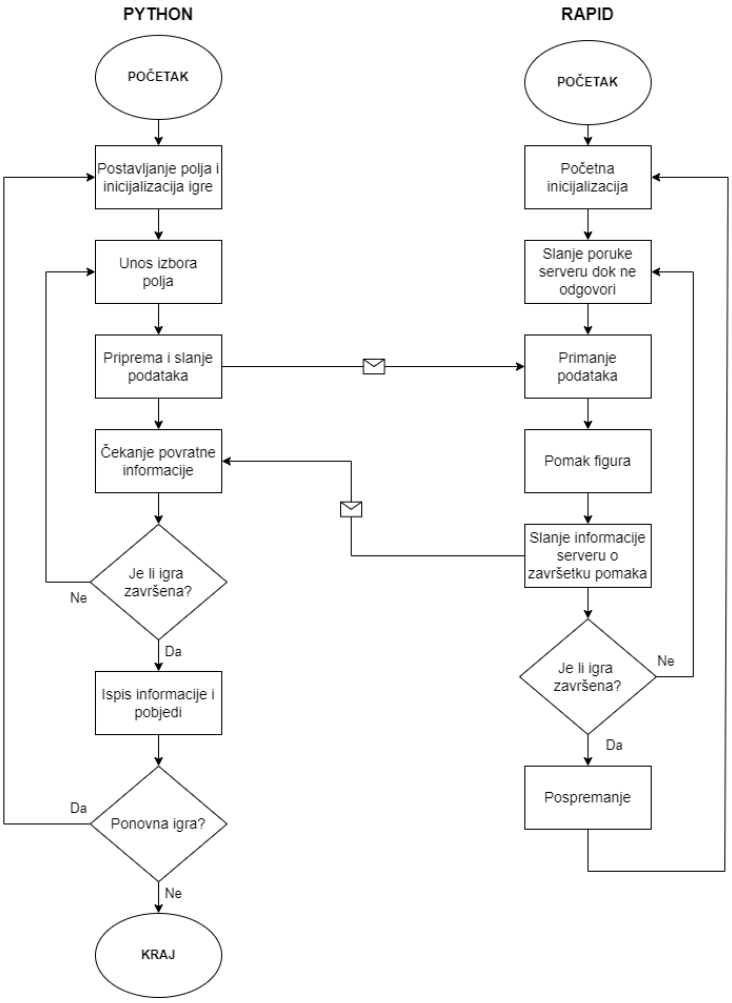
Korištenje više programskih jezika ne bi bilo moguće bez neke vrste komunikacije između njih, odnosno između više programa koji zasebno izvršavaju svoje programske kodove. Komunikacija služi za razmjenu podataka i slanja povratnih informacija čija kombinacija omogućava sinkroniziran rad između više programskih jezika i koordinirani tok rada. Bez komunikacije između programskih jezika gubi se najveća prednost korištenja više jezika i dolazi do neefikasnog toka rada sustava.

U ovome radu korišteni su programski jezici Python i RAPID za uzmi i postavi (engl. *Pick and place*) radnu operaciju. Točnije za uzmi i postavi operaciju odabrana je igra „križić kružić“ koja omogućuje rad s više radnih objekata i više pozicija na koje je potrebno postavljati figure. Python programski jezik korišten je za unos informacije od strane korisnika. Nakon dobivene informacije Python prolazi kroz algoritam i pri tome šalje RAPID programskom kodu potrebne podatke i daje korisniku vizualnu povratnu informaciju u terminalu. Podatke koje šalje Python su podaci o pozicijama koje robotski sustav mora interpretirati kako bi pravilno izvršio radni zadatak. RAPID programski jezik ima zadatak samo primiti informacije, poslati potrebne povratne informacije i izvršiti predodređena gibanja. Kao komunikacijski kanal koristi se internetska komunikacija ili točnije komunikacija putem *socketa*. Iako je radni zadatak u ovome radu relativno lagan on dobro prikazuje način rada s više programskih jezika i prednosti takvog načina izvršenja radnog zadatka.

2. Opis zadatka i radne okoline

Kao što je navedeno u uvodu zadatak je uzmi i postavi operacija u obliku igre „križić kružić“. Python kod programiran u softveru VS *Code* će poslužiti kao algoritam za igru i imati ulogu serverske strane komunikacije. Zadatak RAPID koda, programiranog u softveru RobotStudio, je postavljanje okoline, potrebnih elemenata poput koordinatnih sustava (engl. *workobjecta*) i točaka putanje (engl. *targeta*) koji služe kao koordinate na kojima se nalaze ili na koje je potrebno postaviti figure koje služe kao vizualni prikaz igre.

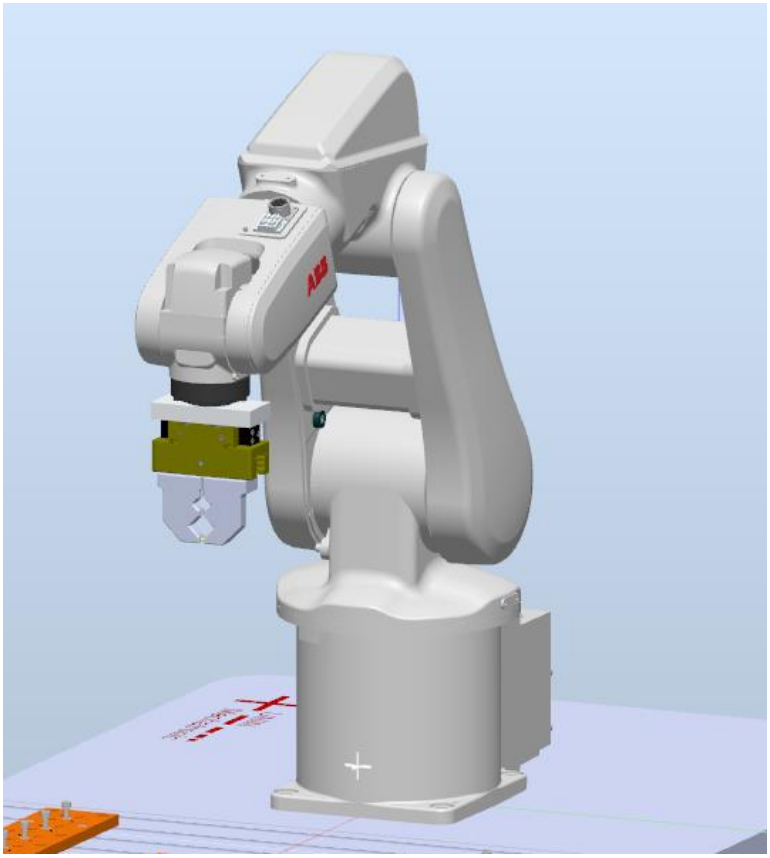
Cijeli će proces (slika 2.1) započeti „crtanjem“ polja za igranje u terminalu i ispisati poruku koja objašnjava korisniku kako nastaviti, odnosno poziva ga na unos odabira polja. Nakon što korisnik potvrdi svoj unos program kreće dalje i šalje podatke RAPID kodu i nakon dobivene povratne informacije nastavlja dalje. U projektu je potrebno da RobotStudio primi potrebne podatke za izvršavanje zadatka. Ti podaci su: koji igrač je napravio potez, koje je odabrano polje i je li taj potez zadnji potez igre. Kada petlja dođe do kraja program crta trenutno stanje polja i proces se ponavlja.



Slika 2.1 Pojednostavljeni dijagram toka procesa

2.1. Robot i kontroler

U svrhu ovog rada korišten je robot IRB 120 tvrtke ABB (Slika 2.2). Taj robot posjeduje 6 osi, maksimalni doseg od 0.58 metara i nosivost do 300 grama [1]. Njegova kompaktnost i aluminijska struktura rezultira manjom težinom što omogućuje montažu gotovo svugdje. Koristi se u velikom broju industrija u zadacima u kojima nije potrebna velika nosivost ili dugačak doseg. To može uključivati zadatke od ispitivanja i kontrole do montaže sklopova sastavljenih od dijelova male težine. Robot ne bi mogao raditi bez potrebnog kontrolera. IRB 120 koristi kontroler IRC5 *Compact* kojeg također proizvodi tvrtka ABB. Standardni paket omogućuje 16 digitalnih ulaza i 16 digitalnih izlaza, ali je moguće i kupiti dodatne pakete koji omogućavaju dodatne opcije. Nažalost IRC5 *Compact* u laboratoriju sveučilišta nema paket PC Interface 616-1 koji omogućuje TCP/IP komunikaciju, te će stoga ispravnost programa biti prikazana unutar simulacije softvera RobotStudio.



Slika 2.2 Robotska ruka i krajnji efektor u razvojnom okruženju RobotStudio

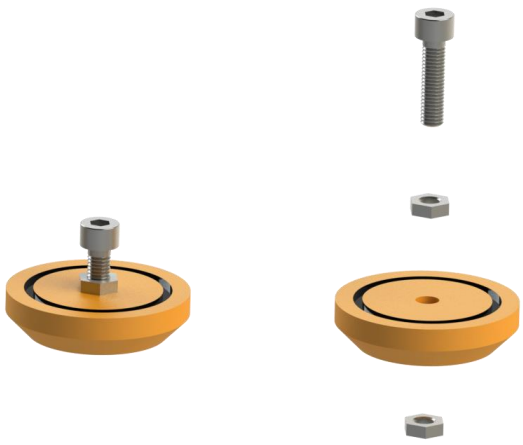
Kako bi robotski sustav mogao obaviti određeni zadatak potreban mu je krajnji efektor s kojim može manipulirati okolinu. Baza krajnjeg efektora je univerzalna pneumatska hvataljka PGN-P 80-1 tvrtke Schunk. Korak svake čeljusti je 8 mm sa silom zatvaranja od 550 N. Prsti čeljusti koji

hvataju figure nisu kupljeni od ni jednog većeg proizvođača već su izrađeni tehnologijom 3D ispisa na fakultetu. Napajanje zraka za zatvaranje i otvaranje krajnjeg efektora dobiva se iz konektora na samom tijelu robota na koji zrak dolazi iz cijevi koja su provedene kroz za to određene kanale u samome kućištu industrijskog robota.

2.2. 3D modeli

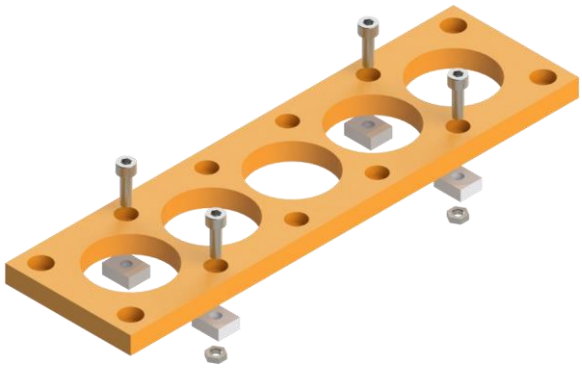
Prvotni plan za ovaj projekt bio je fizički izraditi postolja i figure kako bi se moglo u stvarnosti prikazati funkcionalnost, te su stoga svi predmeti koji su se trebali izraditi 3D ispisom izmodelirani pomoću 3D CAD softvera. Cijela se okolina robotskog sustava sastoji od postolja za igranje, postolja za figure i samih figura uz dodatne elemente koji služe za zadaće poput pričvršćivanja i slično.

Figure za igranje (slika 2.3) sastoje se od nekoliko elemenata. Jedan od dijelova je baza konusnog oblika, vanjskog promjera 39.8 mm, na kojoj se može nalaziti znak križić ili kružić koji je udubljen u samo tijelo baze. Tijelo baze figure visoko je 10 mm. Na sredini se nalazi prolazna rupa od 5.2 mm za vijak ISO 4762 M5 x 20 – 20N. Glava tog vijka ima ulogu mjesta za hvatanje figure. Ukupna visina nakon spajanja svih dijelova iznosi 25 mm. Figura također ima dvije matice ISO 4036 M5 – C koje su odabrane zbog njihove male visine. Baza na sebi posjeduje utor za donju maticu dok je gornja leži na gornjoj plohi tako da se može zategnuti.



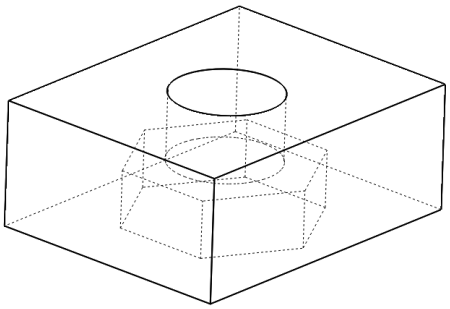
Slika 2.3 Figura za igranje

Postolje za figure (Slika 2.4) dimenzija 70x250x10 mm sastoji se od 5 mjesta za figure promjera 40 mm što je za 0.2 mm veće od promjera baze figura kako bi se izbjegli potencijalni problemi zbog mogućih pogrešaka kod 3D ispisa. Postoji 8 utora na samom komadu koji su namijenjeni za vijke ISO 4762 M5 koji su međusobno razmaknuti za 45 mm kako bi se mogli pričvrstiti za stol koji se nalazi u laboratoriju Sveučilišta Sjever.



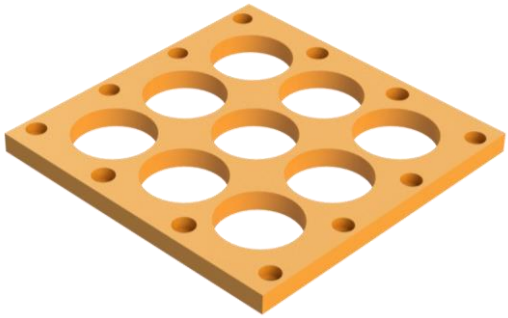
Slika 2.4 Ploča za figure

Također postoje i četiri utora koji su namijenjeni za vijke ISO 4762 M6, međusobno udaljeni 50 mm, kako bi se to isto postolje moglo pričvrstiti za pomičnu ploču koja se nalazi odmah kraj stola robota u laboratoriju Sveučilišta Sjever. Pričvršćivanje ploče na stol je zamišljeno tako da se koriste vijci ISO 4762 M5 x 16 – 16C koji prolaze kroz prolazne rupe u podmetačima za fiksiranje (Slika 2.5) koji su prizmatičnog oblika dimenzija 12x15x5,9 mm koji na sebi imaju prolaznu rupu za vijak i utor za maticu ISO 4036 M5 – C.



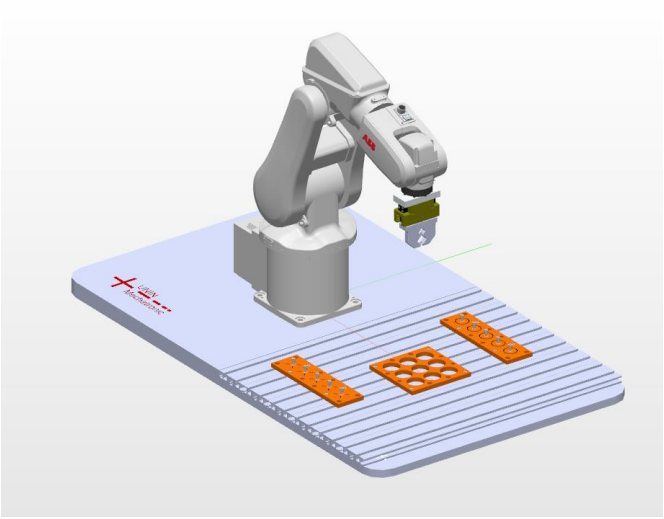
Slika 2.5 Podmetači za fiksiranje s prikazanim nevidljivim bridovima

Postolje za igranje (Slika 2.6) dimenzija 160x160x10 mm sastoji se od 9 provrta za figure koji su kao i postolje za figure promjera 40 mm. Osim tih provrta također postoje i utori za vijke. Na ploči se nalazi osam utora za vijke ISO 4762 M5 za pričvršćivanje ploče za stol i četiri utora za ISO 4762 M6 za pričvršćivanje ploče za pomičnu ploču kraj stola s robotom u laboratoriju Sveučilišta Sjever. Razmaci između tih utora za vijke su isti kao i kod postolja za figure, odnosno razmak od 45 mm za ISO 4762 M5 i razmak od 50 mm za ISO 4762 M6. Pričvršćivanje za stol izvedeno je na isti način kao i ploča za figure.



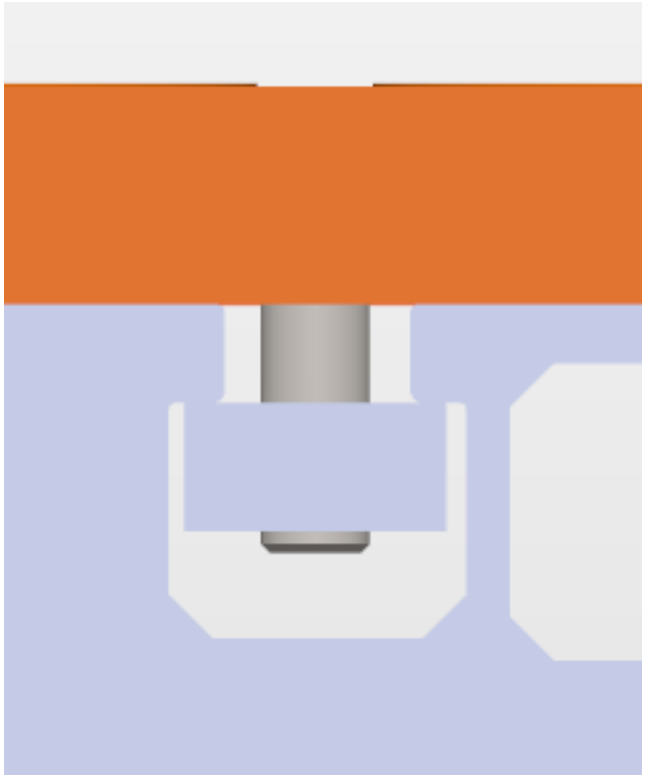
Slika 2.6 Ploča za igranje

Za igru su potrebne dvije ploče za figure od kojih će jedna biti za „križić“ figure, a druga za „kružić“ figure. Uz to potrebna je jedna ploča za igranje na kojoj će biti vizualni prikaz trenutnog stanja igre i na kraju potrebne su figure kojima će se prikazivati stanje igre. Bit će potrebno pet figura s oznakom „križić“ i dodatnih pet figura s oznakom „kružić“.



Slika 2.7 Postavljeni elementi za igru u programskom okruženju RobotStudio

Zbog lakše manipulacije 3D modelima u programu RobotStudio prvo su bile sastavljene ploče s podmetačima i vijcima, te su zatim ti spojeni modeli bili umetnuti u RobotStudio.



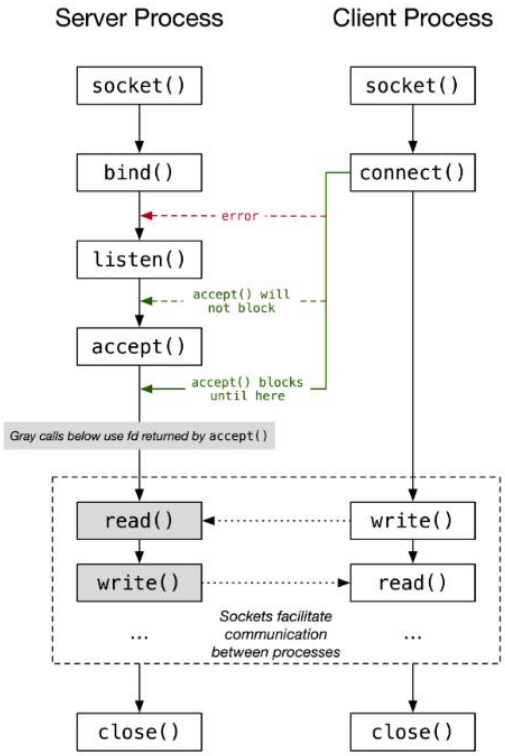
Slika 2.8 Približeni prikaz pričvršćivanja ploče na stol

3. Komunikacija

S obzirom na to da je potrebna komunikacija između dva programska koda, odnosno između dvije aplikacije potrebno je poslati podatke kako bi se program izvršio kako je to zamišljeno. Komunikacija se izvodi tzv. *socketima*. *Socket* je krajnja točka dvosmjerne komunikacije između dva uređaja, odnosno dvije aplikacije, koji se nalaze na istoj mreži [2]. S porastom korištenja interneta poraslo je i korištenje *socketa* za komunikaciju, te se danas koriste za brojne usluge poput pregledavanja weba, razmjena datoteka i sl.

Postoji više vrsta *socketa*, ali dvije vrste su najčešće. Prva od njih je *Stream socket* koja koristi TCP protokol za prijenos podataka. Glavne karakteristike tih *socketa* su pouzdana komunikacija kod koje u slučaju pogreške pošiljatelj ponovno šalje podatke i pošiljatelj paketima podataka koji se šalju pridodaju brojeve koji označavaju njihov redoslijed. Druga vrsta su *Datagram socketi* kod kojih se podaci ne šalju ponovo u slučaju pogreške i paketi podataka mogu doći u bilo kojem redoslijedu. Stoga programi koji koriste tu vrstu *socketa* moraju sadržavati vlastitu obradu grešaka, te također moraju samostalno odrediti redoslijed spajanja paketa podataka u cjelinu [3].

Cijeli proces komunikacije sa *socketima* (Slika 3.1) relativno je jednostavan. U početku na strani poslužitelja i klijenta je potrebno definirati *sockete*. To se radi funkcijom .*socket*() gdje također definiramo koju vrsta *socketa* želimo. Sa strane poslužitelja potrebno je proći još nekoliko koraka dodatnim funkcijama.



Slika 3.1 Dijagram toka socket programiranja [4]

Jedna od tih je funkcija *.bind*() koja povezuje *socket* s određenom adresom i brojem porta. Sljedeća je *.listen*() kojom *socket* „sluša“ mrežu kako bi detektirao zahtjeve za konekciju od strane klijenta na mreži. Njome također možemo zadati maksimalni broj spojenih klijenata prije nego se automatski počnu odbijati novi zahtjevi za konekciju. Zadnja funkcija specifična za stranu poslužitelja je *.accept*() koja prihvaća konekciju kod detektiranog zahtjeva od strane klijenta.

Klijent šalje zahtjev za detekciju funkcijom *.connect*() koja pokreće tzv. *handshake* kojim se potvrđuje da podaci poslani od poslužitelja mogu stići do klijenta i obratno. Nakon tih koraka moguće je slanje podataka koje želimo. To se radi funkcijama *.send*() koja šalje podatke i *.recv*() kojom se podaci primaju. Kao što je vidljivo na dijagrama funkcije *.send*() i . *recv*() se koriste naizmjenično.

Nakon izmijenjenih podataka potrebno je prekinuti vezu što se radi funkcijom *.close*() kao što je vidljivo na dijagramu funkcija *.close*() potrebna je i na strani poslužitelja i na strani klijenta. Iako su u ovome poglavlju opisane funkcije čija se sintaksa koristi u Python programskom jeziku drugi jezici imaju vrlo slične funkcije koje se koriste na isti način [4].

4. RobotStudio okolina i RAPID kod

4.1. Okolina

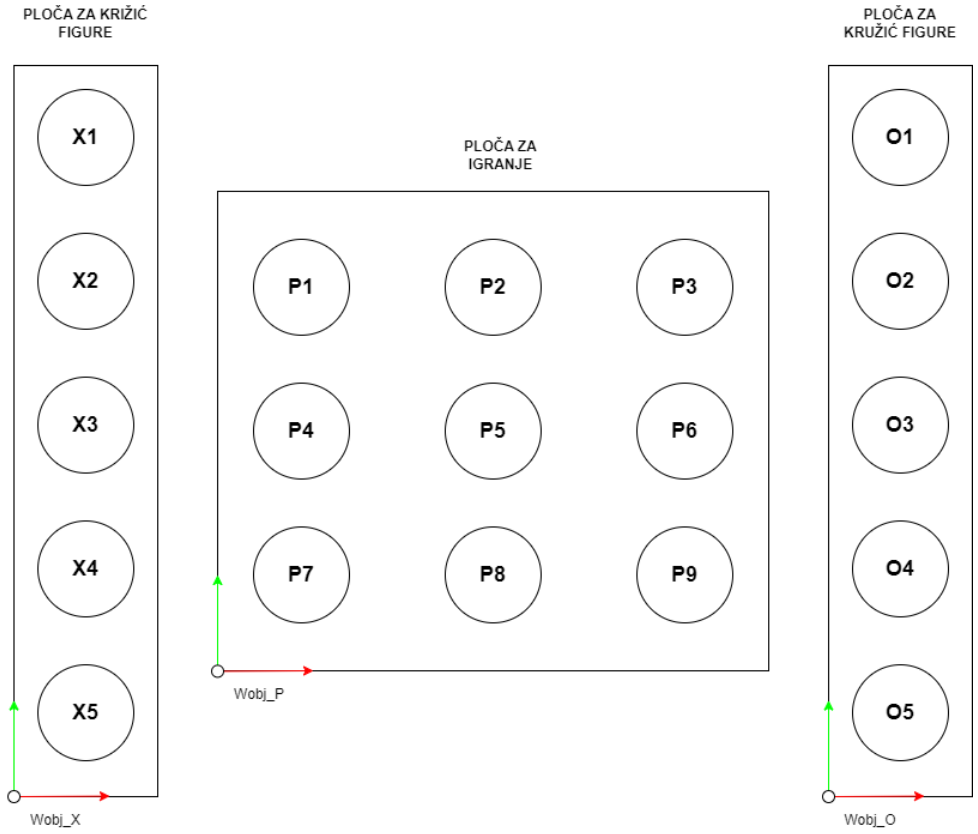
Prije pisanja koda potrebno je postaviti okolinu. Softver RobotStudio ima mogućnost postavljanja vlastitih 3D modela u virtualno okruženje. Postavljanjem virtualnog okruženja omogućuje se lakše praćenje rada programa tijekom simulacije izvođenja programskog koda, odnosno može se vidjeti kako će se fizički modeli pomicati u stvarnosti tokom radnog ciklusa.

Postavljanje okoline započinje umetanjem 3D modela unutar virtualnog okruženja. Nakon što RobotStudio učita 3D model postavlja njegovo ishodište u ishodište koordinatnog sustava virtualnog okruženja. To često nije povoljno te je potrebno raznim alatima koji postoje unutar programskog okruženja RobotStudio pomaknuti i postaviti 3D modele u položaj koji bolje odgovara odvijanju programa. Jedna od opcija koju ima RobotStudio je prikaz radnog prostora robota što omogućuje lakše postavljanje 3D modela na mjesto u prostoru koje industrijskog robota može dosegnuti.

Slika 4.1 Okolina s prikazom radnog prostora

Prije nego se krene s programiranjem poželjno je postaviti dodatne objekte unutar okruženja koji će pomoći kod pisanja programa i oblikovati sam izgled programa. Jedan od tih objekata je koordinatni sustav (engl. *Workobject).* *Workobject* je koordinatni sustav koji opisuje poziciju radnog komada [5]. U ovome projektu svaka ploča ima svoj *workobject* za koji su vezane točke

putanje robota, odnosno *Robot target* objekti. Na taj se način ostvaruje mogućnost razmještanja fizičkih dijelova projekta uz nastavak ispravnog rada cijelog sustava. Ukoliko bi se koristio jedan workobject kod razmještanja, npr. Ploče za križić figure, morale bi se ažurirati lokacije svih figura dok se kod korištenja zasebnih *workobjecta* mijenja samo pozicija tog *workobjecta* za koji su vezane potrebne točke putanje. Za potrebe ovog projekta koriste se tri zasebna koordinatna sustava. Svaka fizička ploča ima za sebe vezan jedan od koordinatnih sustava za koje su još vezane točke putanje, odnosno točke u prostoru u koje se industrijski robot pozicionira.



Slika 4.2 Shematski prikaz položaja *workobjecta* i *targeta* (točaka)

Za pravilno funkcioniranje programa osim točaka putanje na shematskom prikazu potrebne su još dodatne točke „T_Igrac“ i „T_Polje“ koji su za razliku od ostalih točaka varijable, odnosno mogu im se mijenjati vrijednosti. One služe kao privremene točke kojima pridodajemo vrijednosti stvarnih točaka kako ne bi trebali pisati posebne naredbe za svaku potrebnu točku već dio programa za pomak koristi samo te dvije točke putanje kojima pridodajemo vrijednosti stvarnih točaka u koje se želimo pozicionirati. Osim tih točaka definira se još jedan proizvoljan *target* „Home“ koji je položaj iznad ploče u koji robot dolazi nakon završetka poteza.

Nakon postavljanja svih *targeta* i *workobjecta* u virtualnom okruženju potrebno je te podatke prenijeti u sam programski kod. Za to postoji opcija unutar Robot Studia „*Synchronize to RAPID*“ koja uzima podatke iz virtualnog okruženja i generira deklaracije tih objekata. Nakon što RAPID ima podatke o svim potrebnim objektima može se započeti s pisanjem programskog koda.

4.2. RAPID programski kod

Dio projekta koji je pisan u RAPID programskom kodu ima nekoliko funkcija. Najbitnija funkcija je ostvarivanje pomaka figura što služi kao fizički prikaz trenutnog stanja igre. Vezano uz komunikaciju RAPID kod ima ulogu klijenta, odnosno on je taj koji mora poslati zahtjev za početak komunikacije. Uz to RAPID kod također ima još nekoliko aspekata poput zapisa na kojoj se poziciji nalazi koja figura i logiku koja interpretira dobivene podatke.

RAPID kod započinje definicijama potrebnih varijabli poput *workobjecta* i *robtargeta* (Kod 4.1). Za rad su potrebna tri *workobjecta*. WObj_X koristi se kao koordinatni sustav za točke na ploči za figure na kojoj se spremaju figure s oznakom „križić“. Analogno tome Wobj_O koristi se kao koordinatni sustav za točke na ploči za figure gdje se spremaju figure s oznakom „kružić“. Wobj_P koristi se za spremanje točaka koje služe kao polja koje igrači mogu izabrati za postavljanje figure na to mjesto. Nakon deklaracije *workobjecta* slijede deklaracije točaka putanje, odnosno *robtargeta*. Točke od O1 do O5 služe kao točke na ploči za figure koja služi za spremanje figura s oznakom „kružić“. Točke od P1 do P9 služe kao točke u koje se postavljaju figure na ploču za igranje. Točke X1 do X5 služe kao točke na ploči za figure gdje se spremaju figure s oznakom „križić“. *Robtarget* Home služi samo kao točka u koju se robot postavlja kako bi došao na sigurnu visinu. Točke T_Igrac i T_Polje služe kao privremene točke koje se mogu mijenjati po potrebi što smanjuje broj linija koda dijela programa zaduženog za pomak robota. Zbog preglednosti neke su linije koda skraćene ili izbačene.

```
PERS tooldata PrihvatnicaHanoi:= [TRUE,[[0,13,118],[1,0,0,0]],…

TASK PERS wobjdata Wobj_X:=[FALSE,TRUE,"",[[455.104,-248.405,10]…
TASK PERS wobjdata Wobj_P:=[FALSE,TRUE,"",[[500,-80.16,10],…
TASK PERS wobjdata Wobj_O:=[FALSE,TRUE,"",[[500.104,176.407,10],…

VAR robtarget Home:=[[80,80,300],…
CONST robtarget O1:=[[35,215,-10],…
CONST robtarget O2:=[[35,170,-10],…
…
CONST robtarget O5:=[[35,35,-10],…
CONST robtarget P1:=[[35,125,-10],…
CONST robtarget P2:=[[80,125,-10],…
…
CONST robtarget P9:=[[125,35,-10],…
CONST robtarget X1:=[[35,215,-10],…
CONST robtarget X2:=[[35,170,-10],…
…
CONST robtarget X5:=[[35,35,-10],…
VAR robtarget T_Igrac:=[[80.16,185,425],…
VAR robtarget T_Polje:=[[80.16,185,425],…
```

Kod 4.1 Deklaracije workobjecta i robtargeta

Nakon što su definirani potrebni objekti za spremanje i manipulaciju točaka programa definiraju se varijable koje su potrebne za komunikaciju (Kod 4.2). Potrebni *socketi* moraju biti deklarirani kao *socketdev* varijable kako bi se mogli koristiti unutar programa. *clientSocket* se koristi za klijenta, dok se *serverSocket* koristi za poslužitelja. Nakon toga deklarirane su tri varijable tipa *string*, odnosno tekstualne varijable. One se koriste za spremanje podataka poslanih od strane Python koda, odnosno od strane poslužitelja. U varijablu Pozicija sprema se odabir polja na ploči za igranje, tj. polje na koje se stavlja figura. Varijabla Igrač služi za spremanje informacije o tome koji je igrač radi taj potez koji je spremljen u varijablu Pozicija. Varijabla Igrač može biti ili „1“ za igrača koji pomiče figure s oznakom „križić“ ili „2“ za igrača koji pomiče figure s oznakom „kružić“. Varijabla Igranje služi za spremanje informacije o tome je li trenutni potez zadnji potez igre. RAPID kod kasnije na temelju toga odlučuje hoće li započeti rutinu koja posprema figure s ploče za igranje na ploču za figure kako bi igra mogla ponovo započeti.

```
VAR socketdev serverSocket;
VAR socketdev clientSocket;
VAR string Pozicija;
VAR string Igrac;
VAR string Igranje;
```

Kod 4.2 Deklaracija varijabli za komunikaciju

Potrebno je također deklarirati varijable koje služe kao pripomoć za određene procese (Kod 4.3). Varijable FiguraX i FiguraO služe za praćenje mjesta s kojeg uzimamo figure s ploče za figure. Njih u programu inkrementiramo tijekom trajanja igre kako bi svaki sljedeći put uzeli sljedeću figuru bez da bespotrebno širimo program. Kako tokom trajanja igre inkrementiramo tako kod rutine pospremanja dekrementiramo te varijable kako bi svaki puta kod pospremanja stavili figure na sljedeće slobodno mjesto. Tim varijablama odmah kod deklaracije pridodajemo vrijednost 0. Njihovu vrijednost kod rutine pospremanja ne stavljamo izravno na nulu već one putem dekremenitranja kroz proces pospremanja same dolaze do vrijednosti nula, te su spremne za novu igru. Iduće varijable služe za zapis koje figure se nalaze na kojoj poziciji na ploči za igranje. *Boolean* varijabla ok i brojčana varijabla potez služe za pretvorbu *stringa* u brojčanu varijablu kako bi se moglo manipulirati *string* polje nazvano zapis. Varijabla zapis je polje (engl. *Array*) koja se koristi za zapis figura na polju. Tijekom programa će se „0“ u polju zamijeniti s „1“ ili “2“ ovisno o tome koji je igrač napravio potez, odnosno koja se figura nalazi na kojem polju ploče za igranje. Kako bi se članovi polja mogli mijenjati potrebno je indeksima pristupiti

određenim dijelovima polja. Ti indeksi moraju biti u brojčanom obliku dok podaci dolaze u *string* obliku, te ih je zbog toga potrebno pretvoriti u brojčani oblik, odnosno u varijablu tipa *num*.

```
VAR num FiguraX :=0;
VAR num FiguraO :=0;
VAR num potez;
VAR bool ok;
VAR string zapis{9}:=["0","0","0","0","0","0","0","0","0"];
```

Kod 4.3 Deklaracija pomoćnih varijabli

Kada su deklarirane potrebne varijable kreće glavna petlja programa koja se nalazi u rutini *Main* (Kod 4.4). Cijela se rutina odvija u beskonačnoj petlji koristeći oznaku „strt“ i naredbu *goto* strt koja vraća *program pointer* na liniju koda s tom oznakom. Korištena je oznaka „strt“ umjesto „start“ jer RAPID kod, odnosno njegov kompajler koristi riječ start za nešto drugo pa ne dopušta njeno korištenje kao oznaku. Odmah nakon oznake nalazi se komanda *Comm* koja govori programu da ide u rutinu s istim nazivom koja će biti opisana kasnije u ovom radu.

```
PROC Main()
     strt:
     Comm;
     IF Igranje = "1" THEN
         Pospremanje;
         MoveJ Home,v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_P;
     ENDIF
     GOTO strt;
ENDPROC
```

Kod 4.4 Main petlja

Nakon izlaska iz te rutine program se vraća u glavnu rutinu i ide na sljedeću naredbu koja provjerava vrijednost varijable Igranje koja mu govori radi li se o zadnjem potezu igre i treba li početi s pospremanjem. Ako je vrijednost te varijable „1“ program ide u rutinu Pospremanje koja će biti opisana kasnije u ovome radu. Kod izlaska iz rutine Pospremanje robot se podiže na sigurnu visinu u poziciju Home i program izlazi iz *if* petlje, te dolazi do naredbe *goto* koja ga zbog oznake „strt“ vraća na početak i cijeli se proces ponavlja. U slučaju da je vrijednost varijable pospremanje različita od „1“ cijela se *if* petlja preskače.

U rutini *Comm* (Kod 4.5) nalazi se dio programa koji je zadužen za komunikaciju s Python kodom, odnosno poslužiteljem. Na početku rutine nalazi se naredba koja pomiče robota u točku Home odnosno podiže prihvatnicu robota na sigurnosnu visinu. Nakon toga slijede naredbe *SocketCreate* koje stvaraju klijent i poslužitelj *sockete* unutar koda kako bi se moglo krenuti s komunikacijom. Također je potrebno dati programu informacije o IP adresi servera i o portu preko kojeg se komunicira putem naredbe *SocketConnect*. Osim IP adrese i porta, naredbi se također može dati podatak o maksimalnom vremenu u sekundama koje program čeka za uspostavu konekcije. U ovome slučaju to je vrijeme podešeno na četiri sekunde. Ako program ne uspije javlja se greška s kodom ERR_SOCKET_TIMEOUT, te se ta greška obrađuje kasnije u programu ako do nje dođe. Nakon uspostave konekcije počinje slanje i primanje podataka.

```
PROC Comm()

    MoveJ Home,v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_P

    SocketCreate clientSocket;
    SocketCreate serverSocket;
    SocketConnect serverSocket, "192.168.0.1", 5000 \Time:=4;

    SocketSend serverSocket \Str:="Zahtjev";
    SocketReceive serverSocket \Str:= Pozicija;
    SocketSend serverSocket \Str:="OK";
    SocketReceive serverSocket \Str:= Igrac;
    SocketSend serverSocket \Str:="OK";
    SocketReceive serverSocket \Str:= Igranje;
    Pomak;
    SocketSend serverSocket \Str:="Zavrseno";

    SocketClose serverSocket;
    SocketClose clientSocket;
```

Kod 4.5 Razmjena podataka

Prvo klijent (RAPID) šalje tekst „Zahtjev“ koji služi samo za sinkronizaciju i početak slanja podataka. Nakon što poslužitelj (PYTHON) primi tekst šalje natrag podatke o tome koje je polje na ploči za igranje izabrano i klijent to sprema u varijablu Pozicija. Dobivanjem i spremanjem podataka o izabranom polju klijent šalje tekst „OK“, te nakon što ga poslužitelj primi šalje podatke o igraču koji je napravio taj potez koji se sprema u varijablu Igrac. Kao i u prijašnjem dijelu koda klijent opet šalje tekst „OK“, te pritom natrag dobiva informaciju o stanju igre nakon poteza što se sprema u varijablu Igranje. Time klijent ima sve potrebne podatke i kreće u rutinu Pomak koja će biti opisana kasnije u ovome radu. Za to vrijeme dok klijent obavlja pomicanje figura poslužitelj čeka informaciju o završetku pomaka. Nakon što je pomicanje figure završeno klijent šalje tekst „Zavrseno“ što daje poslužitelju signal da može dalje nastaviti s radom. Time završava razmjena podatka i zatvara se komunikacijski kanal naredbama *SocketClose* za *serverSocket* i *clientSocket*.

Ako *SocketConnect* ne uspije uspostaviti konekciju javlja se greška ERR_SOCKET_TIMEOUT. Ukoliko program pet puta za redom ne uspije uspostaviti vezu program prekida s radom. Kako bi se to izbjeglo u programu postoji nekoliko linija koda (Kod 4.6) koje resetiraju broj pokušaja kako ne bi došli do preranog izlaska i kraja programa. Ukoliko dođe do greške ERR_SOCKET_TIMEOUT program ulazi u iz petlju, resetira broj pokušaja i vraća se ponovno na mjestu na kojem je bio kada se javila greška.

```
      ERROR
      IF ERRNO = ERR_SOCK_TIMEOUT THEN
            ResetRetryCount;
            RETRY;
      ENDIF

 ENDPROC
```

Kod 4.6 Obrada grešaka

Kada klijent, nakon primljenih podataka o odabranom polju i igraču, primi podatke o stanju igre nakon poteza može krenuti s pomicanjem figura što se nalazi u posebnoj rutini Pomak. Na početku rutine provjerava se vrijednost varijable Igrac kako bi se odredilo treba li pomaknuti figure „križić“ ili „kružić“ .Koje figure će se pomaknuti određeno je *if* petljama koje provjeravaju vrijednost varijable Igrac i vrijednost varijable FiguraX, odnosno vrijednost varijable FiguraO. Ukoliko vrijednost varijable Igrac iznosi „1“ ulazi se u prvu *if* petlju (Kod 4.7) u kojoj se prije svega inkrementira, odnosno poveća za jedan, varijabla FiguraX. Kada program prvu puta uđe u tu petlju poveća se vrijednost te varijable s nula na jedan što se kasnije koristi u *Test-Case* strukturi.

```
PROC Pomak()
    IF Igrac = "1" THEN

        Incr FiguraX;

        TEST FiguraX
            CASE 1: T_Igrac:=X1;
            CASE 2: T_Igrac:=X2;
            CASE 3: T_Igrac:=X3;
            CASE 4: T_Igrac:=X4;
            CASE 5: T_Igrac:=X5;
        ENDTEST

    ENDIF
```

Kod 4.7 Određivanje točke uzimanja križić figure

Test-Case, češće u drugim programskim jezicima *Switch-Case,* struktura provjerava (*test*) vrijednost neke varijable i na temelju toga izvodi naredbe zapisane u jednim od mogućih slučajeva (*case*). Ako na primjer u programu prvi puta ulazimo u tu petlju FiguraX će, kod dolaska do dijela s *Test-Case* strukturom, imati iznos jedan. Na temelju toga će program otići u liniju koda *CASE 1* koja će privremenoj varijabli T_Igrac preslikati vrijednost *robtarget* varijable X1 koja je u stvari točka na kojoj se nalazi prva figura s oznakom „križić“ na ploči za figure. Na taj se način značajno skraćuje broj potrebnih linija koda koju govore robotu u koju se poziciju mora pomaknuti da uzme figuru s oznakom kružić. Kada program sljedeći put dođe u tu petlju vrijednost varijable FiguraX poprimit će vrijednost dva, što će kod dolaska do *Test-case* strukture uputiti program da ide u liniju koda *CASE 2* što će varijabli T_Igrac pridodati vrijednost X2 i tako do kraja dok igra ne završi.

Analogno tome *if* petlja u koju se ulazi kada je igrač s oznakom kružić izveo potez (Kod 4.8) radi na isti način. Jedine promjene su imena nekih varijabli. Umjesto varijable FiguraX koristi se varijabla FiguraO i umjesto točaka od X1 do X5 koriste se točke od O1 do O5.

```
IF Igrac = "2" THEN

    Incr FiguraO;

    TEST FiguraO
        CASE 1: T_Igrac:=O1;
        CASE 2: T_Igrac:=O2;
        CASE 3: T_Igrac:=O3;
        CASE 4: T_Igrac:=O4;
        CASE 5: T_Igrac:=O5;
    ENDTEST

ENDIF
```

Kod 4.8 Određivanje točke za uzimanja kružić figure

Kada je odabrana lokacija s koje će se uzeti figura potrebno je odrediti lokaciju na koju će se postaviti, odnosno na koje je polje na ploči za igranje potrebno postaviti tu figuru. Za taj se zadatak, kao i za odabir lokacije uzimanja figure, koristi *Test-Case* struktura (Kod 4.9) jedino što ovaj put nije potrebna *if* petlja i provjerava se iznos varijable Pozicija. Ovisno o vrijednosti varijable Pozicija varijabli T_Polje pridodaje se vrijednost *robtargeta* koje su zapravo točke na ploči za igranje. Ako je na primjer u varijablu Pozicija spremljena vrijednost „2“ tada će se, kada program dođe do te *Test-Case* strukture, varijabli T_Polje pripisati vrijednost *robtargeta* P2.

```
TEST Pozicija
     CASE "1": T_Polje:=P1;
     CASE "2": T_Polje:=P2;
     CASE "3": T_Polje:=P3;
     CASE "4": T_Polje:=P4;
     CASE "5": T_Polje:=P5;
     CASE "6": T_Polje:=P6;
     CASE "7": T_Polje:=P7;
     CASE "8": T_Polje:=P8;
     CASE "9": T_Polje:=P9;
ENDTEST
```

Kod 4.9 Određivanje točke spuštanja figure za igranje

Pridodavanjem vrijednosti varijablama T_Igrac i T_Polje program posjeduje sve potrebne informacije za pomicanje figura. Dio programa za pomicanje kreće s provjerom koji igrač radi pokret koristeći *if* petlje (Kod 4.10). Ukoliko je igrač s oznakom križić napravio potez program ulazi u tu petlju. Prva naredba služi za pozicioniranje prihvatnice iznad figure.

```
IF Igrac = "1" THEN
     MoveJ Offs(T_Igrac,0,0,60),v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_X;
     Otvori_Griper;
     MoveL Offs(T_Igrac,0 0,22.5),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_X;
     Zatvori_Griper;
     MoveL Offs(T_Igrac,0,0,60),v100,z100,PrihvatnicaHanoi\WObj:=Wobj_X;
     WaitRob \InPos;
ENDIF
```

Kod 4.10 Uzimanje križić figure s ploče za figure

Nakon toga slijedi naredba za pokretanje rutine Otvori_Griper (Kod 4.11). U njoj bi, u stvarnosti, kontroler robotskog sustava dao električni signal koji otvara prihvatnicu. S obzirom na to da u ovome radu sve ostaje u simulaciji rutina za otvaranje prihvatnice ima samo naredbu koja čeka s izvođenjem dok robot ne dođe potpuno u poziciju, postavlja vrijednost virtualnog izlaznog signala DO_1 u logičku nulu i pričeka jednu sekundu da se otvori prihvatnica što je dovoljno za potrebe simulacije. Poslije otvaranja čeljusti prihvatnice robot se može spustiti na visinu na kojoj može uhvatiti figuru. Kada je u toj poziciji program ulazi u rutinu Zatvori_Griper (Kod 4.12) koji je gotovo identičan rutini za otvaranje čeljusti prihvatnice osim što umjesto da postavlja vrijednost virtualnog signala DO_1 u logičku nulu, ta ga rutina postavlja u logičku jedinicu što zatvara čeljusti prihvatnice. Nakon zatvaranja čeljusti prihvatnice industrijski se robot ponovno podiže na visinu u kojoj prilazi figuri prije hvatanja. Na samome kraju *if* petlje je naredba koja govori programu da pričeka dok robot u potpunosti ne dođe u zadanu poziciju i stane.

```
PROC Otvori_Griper()
     WaitRob \InPos;
     Set DO_1;
     WaitTime 1;
ENDPROC
```

Kod 4.11 Rutina za otvaranje čeljusti prihvatnice

```
PROC Zatvori_Griper()
     WaitRob \InPos;
     Reset DO_1;
     WaitTime 1;
ENDPROC
```

Kod 4.12 Rutina za zatvaranje čeljusti prihvatnice

U slučaju da je igrač s oznakom kružić napravio trenutni potez uzimanje figure obavlja druga *if* petlja koja je gotovo identična prvoj (Kod 4.13). Taj je dio programa bilo potrebno razraditi s dvije if petlje zbog više *workobjecta* koji se koriste. Može se vidjeti kako se točka koja se koristi, T_Igrac, ostaje ista u obje petlje, ali *workobject* koordinatnim sustavima ne može se manipulirati kao što se to može s *robtargetima*.

```
IF Igrac = "2" THEN
    MoveJ Offs(T_Igrac, 0, 0, 60),v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_O;
    Otvori_Griper;
    MoveL Offs(T_Igrac, 0, 0, 22.5),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_O;
    Zatvori_Griper;
    MoveL Offs(T_Igrac, 0, 0, 60),v100,z100,PrihvatnicaHanoi\WObj:=Wobj_O;
    WaitRob \InPos;
ENDIF
```

Kod 4.13 Uzimanje kružić figure s ploče za figure

U tim dijelovima programa gdje se uzima figura za igranje koriste se dvije vrste naredbi za pomicanje alata u točku. Te su naredbe *MoveJ* i *MoveL*. Razlika je u putanji koju alat, odnosno robot, prati kod pomaka između prethodne i sljedeće točke. *MoveJ* naredba između dvije točke putuje po proizvoljnoj, optimalnoj putanji koja se opisuje nekom krivuljom u prostoru, te se ona u programu koristi na mjestima gdje ne može doći do kolizije s nekim drugim predmetima. Točnije u ovome programu *MoveJ* se koristi u prilazu iznad figura gdje nema drugih predmeta. Na mjestima gdje postoji mogućnost kolizije, kao što je prostor netom oko ploče s figurama ili ploče za igranje, koristi se *MoveL* naredba za pomak kojom putanja između dvije točke poprima oblik

pravca u prostoru što smanje mogućnost kolizije s drugim objektima. Takav pristup programiranju putanje koristi se i u ostatku programskog koda gdje je potreban pomak.

U ovim se primjerima također može vidjeti korištenje *offset* (*Offs*) funkcije kod pomaka. S obzirom na to da je potrebno manevriranje oko figure bez te naredbe bilo bi potrebno koristiti više točaka što smanjuje preglednost programa. Funkcijom *offs* može se napraviti pomak u neku točku koja je za određen broj milimetara u tri prostorne osi zamaknuta od već definirane točke. Ta funkcija posebno dolazi do izražaja u izrazito kompleksnim programima s velikim brojem pomaka i točaka.

Povodom uzimanja figure i podizanja na sigurnu visinu program kreće s postavljanjem figure na ploču za igranje (Kod 4.14). U početku se prihvatnica s figurom pozicionira na sigurnoj visini iznad pozicije na koju se želi postaviti. Kada je to obavljeno program čeka da industrijski robot u potpunosti stane. Nakon toga kreće spuštanje s vrlo malom brzinom do točke u kojoj se ispušta figura. Dolaskom u tu točku kreće rutina za otvaranje čeljusti prihvatnice i podizanje na sigurnu visinu iznad ploče nakon otvaranja. Na kraju svega se izvrši naredba koja pozicionira ruke u Home poziciju i time je završeno postavljanje figure na ploču za igranje.

```
MoveJ Offs(T_Polje, 0, 0, 60),v1000,z0,PrihvatnicaHanoi\WObj:=Wobj_P;
WaitRob \InPos;
MoveL Offs(T_Polje, 0, 0, 25),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_P;
Otvori_Griper;
MoveJ Offs(T_Polje, 0, 0, 60),v100,z0,PrihvatnicaHanoi\WObj:=Wobj_P;
MoveJ Home,v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_P;
```

Kod 4.14 Postavljanje figure na ploču za igranje

U ovim se primjerima vidi prednost korištenja privremenih varijabli točaka kojima prepisujemo potrebne vrijednosti. Bez tog načina programiranja bilo bi potrebno onoliko *if* petlji i onoliko dijelova programa za pomak figure koliko je točaka na kojima se nalaze figure i one na kojima se figure postavljaju, a na ovaj se način koriste samo tri.

Iako je pomak figure gotov potrebno je još zapisati taj pomak kako bi program mogao pravilno pospremiti figure nakon završetka igre (Kod 4.15). Taj dio programa započinje pretvorbom *string* varijable Pozicija u brojčanu vrijednost koja se sprema u varijablu potez kako bi se mogla koristiti za indeksiranje polja. RAPID programski jezik također zahtjeva *boolean* varijablu kod pretvorbe, te je iz tog razloga u kodu varijabla ok koja se ne koristi ni za što drugo. S brojčanom varijablom koja predstavlja odabrano polje na ploči za igranje moguć je zapis poteza. Varijabla zapis je jednodimenzionalno *string* polje s devet elemenata koji su na samome početku programa postavljeni kao „0“. Članovi od jedan do devet predstavljaju stanja na fizičkoj ploči na točkama od P1 do P9. Drugom linijom koda za praćenje poteza član koji ima indeks koji je jednak brojčanoj

varijabli potez mijenja svoju vrijednost u vrijednost koju ima varijabla Igrac. Ako, na primjer, igrač s oznakom križić, čime nakon komunikacije varijabla Igrac poprima vrijednost „1“, odabere polje P5 tada se u varijabli zapis, član s indeksom pet mijenja u vrijednost varijable Igrac, odnosno „1“. Drugi bi slučaj bio da, hipotetski, igrač s oznakom kružić, čime varijabla Igrac ima vrijednost „2“, izabere polje P1 tada se u varijabli zapis član s indeksom jedan mijenja u „2“. Ukoliko polje nije odigrano u varijabli zapis član koji predstavlja to polje zadržava vrijednost „0“.

```
ok := StrToVal(Pozicija, potez);
zapis{potez}:=Igrac;
```

Kod 4.15 Zapis odigranog poteza

U slučaju da je trenutni potez zadnji potez igre, odnosno ako je stanje varijable igranje jednako „1“ što je provjereno *if* petljom u glavnoj rutini Main, program ulazi u rutinu Pospremanje. Cijela se rutina sastoji od *for* petlje koja prolazi kroz devet iteracija povećavajući svaki puta internu varijablu i za 1 (Kod 4.16). Varijabla i koristi se odmah u liniji koda nakon toga za pristup pojedinim članovima polja zapis, točnije program ulazi u *if* petlju ako je vrijednost trenutno provjeravanog člana polja različita od „0“. Program će do kraja *for* petlje proći kroz sve članove u polju i na taj način provjeriti nalazi li se na ploči za igranje figura i ako se nalazi može također odrediti koju oznaku figura ima na temelju vrijednosti člana polja.

```
PROC Pospremanje()
        FOR i FROM 1 TO 9 DO
            IF zapis{i} <> "0" THEN
```

Kod 4.16 For petlja rutine

U slučaju da je jedna od članova polja s indeksom i različit od „0“ program prvo na temelju varijable i bira polje s kojeg će se uzimati figura (Kod 4.17). Ako je na primjer član s indeksom tri polja zapis različit od „0“, tada program *robtarget* varijabli T_Polje pridodaje vrijednost *robtargeta* P3. Nakon što je izabrano polje na ploči za igranje s koje će se uzimati figura potrebno je odrediti mjesto na koje će se pospremiti ta figura na ploči za figure. Za to se koriste *if* petlje koje provjeravaju koju oznaku ima figura na tom polju, odnosno provjerava se član polja zapis. U slučaju da član polja koji se provjerava ima vrijednost „1“, tada program zna da se radi o figuri s oznakom križić (Kod 4.17). Analogno tome u slučaju da vrijednost tog člana iznosi „2“ program zna da se radi o figuri s oznakom kružić (Kod 4.18).

```
TEST i
    CASE 1: T_Polje:= P1;
    CASE 2: T_polje:= P2;
    CASE 3: T_polje:= P3;
    CASE 4: T_polje:= P4;
    CASE 5: T_polje:= P5;
    CASE 6: T_polje:= P6;
    CASE 7: T_polje:= P7;
    CASE 8: T_polje:= P8;
    CASE 9: T_polje:= P9;
ENDTEST
```

Kod 4.17 Biranje polja s kojeg se uzima figura

Slično kao kod biranja koju figuru će uzeti kod postavljanja figura tijekom igre, program na temelju varijable FiguraX, odnosno FiguraO, bira *robtarget* na ploči za figure. Nakon što program odabere *robtarget* na koji će pospremiti figuru dekrementira, odnosno smanji za jedan, varijablu FiguraX ili FiguraO kako bi iduću iteraciju bio odabran drugi *robtarget,* tj. iduće slobodno mjesto na ploči za figure.

```
IF zapis{i} = "1" THEN

     TEST FiguraX
         CASE 1: T_Igrac:=X1;
         CASE 2: T_Igrac:=X2;
         CASE 3: T_Igrac:=X3;
         CASE 4: T_Igrac:=X4;
         CASE 5: T_Igrac:=X5;
     ENDTEST

     Decr FiguraX;
ENDIF
```

Kod 4.18 Biranje praznog mjesta za pospremanje na ploči za križić figure

```
IF zapis{i} = "2" THEN

    TEST FiguraO
        CASE 1: T_Igrac:=O1;
        CASE 2: T_Igrac:=O2;
        CASE 3: T_Igrac:=O3;
        CASE 4: T_Igrac:=O4;
        CASE 5: T_Igrac:=O5;
    ENDTEST

    Decr FiguraO;
ENDIF
```

Kod 4.19 Biranje praznog mjesta za pospremanje na ploči za kružić figure

Pridodavanjem vrijednosti *robtarget* varijablama T_Polje i T_Igrac program kreće s naredbama za pomak kojima se navodi robot kod pospremanja figura (Kod 4.20). Na početku tog dijela prihvatnica se postavlja u točku koja je iznad figure koja se posprema i otvaraju se čeljusti prihvatnice. Nakon toga prihvatnica se malom brzinom linearno spušta do točke u kojoj lovi figuru. Dolaskom u tu poziciju rutina Zatvori_Griper zatvara čeljusti čime se lovi figura i na posljetku se robotska ruka s figurom podiže na sigurnu visinu iznad stola.

```
MoveJ Offs(T_Polje, 0, 0, 60),v1000,z100,PrihvatnicaHanoi\WObj:=Wobj_P;
Otvori_Griper;
MoveL Offs(T_Polje, 0, 0, 22.5),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_P;
Zatvori_Griper;
MoveL Offs(T_Polje, 0, 0, 60),v100,z0,PrihvatnicaHanoi\WObj:=Wobj_P;
```

Kod 4.20 Uzimanje figure koja se posprema

Slično kao kod uzimanja figura s ploča za figure tijekom postavljanja potrebno je prvo provjeriti if petljama o kojem se igraču radi zbog dva različita koordinatna sustava, odnosno *workobjecta*, koji se koriste za zasebne ploče za figure. Ta se informacija uzima iz polja zapis. Ako je član s indeksom i jednak „1“ tada je potrebno figuru pospremiti na ploču za „križić“ figure (Kod 4.21). Program prvo pomiče figuru iznad mjesta pospremanja i ne nastavlja rad prije nego robotska ruka dođe točno u taj položaj i stane. Nakon toga se malom brzinom spušta na mjesto otpuštanja i rutina Otvori_Griper otvara čeljusti prihvatnice čime se figura otpušta. Otpuštanjem figure prihvatnica se ponovo podiže na sigurnu visinu i time je pospremanje jedne od figura završeno. Suprotno tome bio bi slučaj da član polja zapis ima vrijednost „2“, te tada program mora pospremiti figuru na drugu ploču za figure (Kod 4.22). Program je gotovo identičan pospremanju križić figure uz razliku da se koristi Wobj_O umjesto Wobj_X.

```
IF zapis{i} = "1" THEN
    MoveJ Offs(T_Igrac, 0, 0, 60),v1000,z0,PrihvatnicaHanoi\WObj:=Wobj_X;
    WaitRob \InPos;
    MoveL Offs(T_Igrac, 0, 0, 25),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_X;
    Otvori_Griper;
    MoveL Offs(T_Igrac, 0, 0, 60),v100,z0,PrihvatnicaHanoi\WObj:=Wobj_X;
ENDIF
```

Kod 4.21 Pospremanje križić figura

```
IF zapis{i} = "2" THEN
    MoveJ Offs(T_Igrac, 0, 0, 60),v1000,z0,PrihvatnicaHanoi\WObj:=Wobj_O;
    WaitRob \InPos;
    MoveL Offs(T_Igrac, 0, 0, 25),v50,z0,PrihvatnicaHanoi\WObj:=Wobj_O;
    Otvori_Griper;
    MoveL Offs(T_Igrac, 0, 0, 60),v100,z0,PrihvatnicaHanoi\WObj:=Wobj_O;
ENDIF
```

Kod 4.22 Pospremanje kružić figura

Prije nego program izađe iz for petlje, odnosno uđe u iduću iteraciju, potrebno je obrisati zapis poteza u *string* polju zapis (Kod 4.23). Na taj je način osigurano da na kraju for petlje svi članovi polja zapis imaju vrijednost „0“ čime je ta varijabla spremna za novu igru.

```
zapis{i}:="0";
```

Kod 4.23 Brisanje zapisa poteza

Izlaskom iz rutine za pospremanje figura program je spreman za novu igru. Kroz tu su rutinu sve potrebne varijable vraćene na početne vrijednosti. Program se natrag vraća u Main i pokušava uspostaviti kontakt s poslužiteljem.

5. Python kod

Dio projekta koji je pisan u Python programskom kodu ima uloga logike koja služi za igranje igre križić kružić. U okviru komunikacije s RAPID programskim kodom ima ulogu poslužitelja koji na zahtjev klijenta šalje podatke. Fleksibilnost i velik broj mogućnosti Python programa omogućuje lako programiranje gdje mnoge funkcije i podatkovne strukture omogućuju kompleksne radnje uz relativno mali broj linija koda.

Python kod podijeljen je u dvije datoteke. Glavna datoteka s petljama u kojima se odvija program i datoteka s funkcijama koje se ili često koriste ili zauzimaju puno mjesta, te se njihovim premještanjem u drugu datoteku poboljšava preglednost i urednost glavnog programa. Zbog tih je razloga na početku programa potrebno uvesti funkcije koje se nalaze u pomoćnoj datoteci pod nazivom funkcije (Kod 5.1). Uvezene su funkcije pod nazivima crtanje, provjeri_red, pobjeda i comm koje će biti opisane kasnije u radu. Osim tih funkcija uvezena je i biblioteka os koja služi za manipuliranjem ispisa u konzoli programa.

```
import os
from funkcije import crtanje, provjeri_red, pobjeda, comm
```

Kod 5.1 Uvezene funkcije

Kada su uvezene sve potrebne funkcije kreće glavni program koji se uglavnom sastoji od dvije *while* ugniježđene petlje. Prvi dio vanjske *while* petlje služi za deklaraciju potrebnih varijabli (Kod 5.2). Petlja *while* izvodi se tako dugo dok *boolean* parametar koji joj se dodaje ima vrijednost *true*, odnosno vrijednost logičke jedinice koja se obično dobiva putem nekog logičkog uspoređivanja kao na primjer a<b i tako dugo dok je taj uvjet ispunjen petlja se odvija. U ovome programu *while* petlji je dan samo parametar *True* i nikakva promjena varijabli unutar programa petlje ne može promijeniti taj parametar u *False*, odnosno logičku nulu koja bi rezultirala izlaskom iz petlje. Iz takve se petlje može izaći jedino naredbom *break* što je učinjeno tako da ne dođe do logičke greške programa koja bi prerano prekinula petlju. Program vanjske petlje započinje definiranjem varijable ploca. Varijabla ploca je podatkovna struktura *dictionary* koja je vrlo slična poljima. Svaki element te podatkovne strukture ima svoj pripadajući ključ (engl. *Key*) koji ima ulogu kao indeks u polju koji omogućuje pristup elementima varijable. U ovome programu ključevi su brojevi od jedan do devet koji predstavljaju polja za igranje na ploči. Uz svaki ključ vezan je element tipa *string* koji predstavlja stanje tog polja na ploči. U početku su to brojevi od jedan do devet koji se kroz program mijenjaju u „X“ ili „O“ ovisno o tome koji je igrač napravio taj potez.

```
while True:
    ploca = {1: '1', 2:'2'…, 8: '8', 9:'9'}
    igra = True
    kraj = False
    red = 0
    prosli_red = -1
```

Kod 5.2 Deklaracija varijabli

Osim varijable ploca potrebno je deklarirati još dvije *boolean* varijable i dvije brojčane varijable. Varijabla igra je *boolean* varijabla koju odmah u startu postavljamo kao *true*, te služi za izvršavanje programa unutarnje petlje tako dugo dok ne dođe do kraja igre. Druga *boolean* varijabla je kraj koja služi za određivanje kako je završena igra, odnosno je li igra završila izjednačeno ili pobjedom jednog od igrača. Prva brojčana varijabla red je tipa *integer* i služi za određivanje znaka koji će se upisati, dok druga varijabla prosli_red služi za provjeru je li prošli unos bio ispravan kasnije u programu.

Deklaracijom varijabli ulazi se u unutarnju petlju u kojoj se odvija igra (Kod 5.3). Ta *while* petlja kao parametar ima varijablu igra i postavljajući nju kao *false* na kraju igre prekida izvršavanje programa unutarnje petlje. Unutarnja petlja kreće s naredbom koja briše sav tekst u konzoli programa kako ne bi ometalo igru. Nakon toga koristi se funkcija crtanje iz druge datoteke koja je zaslužena za crtanje ploče u konzoli programa (Kod 5.4). U toj se funkciji pomoću strukture naziva *f-string* crta ploča kojoj su ulazni parametri elementi varijable ploca. Za ispis se koristi *f-string* jer znatno smanjuje broj potrebnih znakova za ispis ploče u konzolu. *F-string* omogućuje znatno lakše ispisivanje kombinacije iznosa varijabli i teksta jer nema potrebe za konverzijom nekih varijabli u *string* koje zatim povezujemo s ostatkom teksta, već se tekst i varijable pišu na način koji će biti sličan konačnom ispisu.

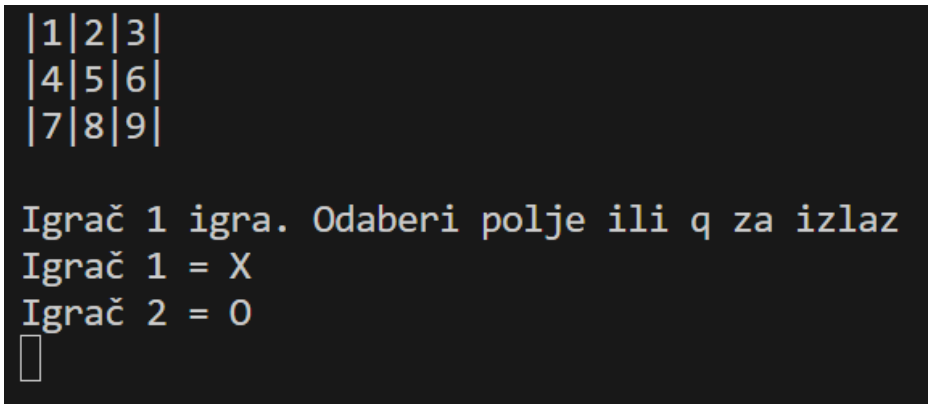
```
while igra:
        os.system ('cls' if os.name == 'nt' else 'clear')

        crtanje(ploca)
        print("Igrač " + str((red%2)+1) + " igra." + " Odaberi
                polje ili q za izlaz")
        print("Igrač 1 = X")
        print("Igrač 2 = O")
```

Kod 5.3 Crtanje ploče u konzoli i ispis informacija

```
def crtanje(ploca):
    ploca = (f"|{ploca[1]}|{ploca[2]}|{ploca[3]}|\n"
             f"|{ploca[4]}|{ploca[5]}|{ploca[6]}|\n"
             f"|{ploca[7]}|{ploca[8]}|{ploca[9]}|\n")
    print(ploca)
```

Kod 5.4 Funkcija za crtanje ploče u konzolu

Nakon crtanja ploče ispisuju se informacije o tome koji je igrač na redu. To je napravljeno tako da se gleda ostatak dijeljenja varijable red s brojem dva. Taj ostatak može biti jedino nula ili jedan što se koristi za određivanje upisa znaka u varijablu ploca. U ovome dijelu programa taj se ostatak koristi za ispis informacije igračima. Ostatku dijeljenja dodaje se broj jedan kako bi se korisnicima dala informacija o tome igra li igrač jedan ili igrač dva. Uz to se još ispisuje dodatna informacija koja govori igračima da je igrač jedan znak križić dok je igrač dva znak kružić. Time se u konzoli korisniku ispisuje izgled ploče i informacije potrebne za igru ili izlaz (Slika 5.1).



Slika 5.1 Ispis u terminalu kod pokretanja programa

Prije nego što se igračima omogući unos poteza potrebno je provjeriti ispravnost prošlog unosa poteza (Kod 5.5). Prvo se provjerava je li iznos varijable prosli_red jednak iznosu varijable red. Kod prvog ulaska u unutarnju petlju taj uvjet neće biti ispunjen jer su u tom trenutku vrijednosti varijabli različiti. Nakon toga varijabli prosli_red se pridodaje vrijednost varijable red. Detekcija netočnog unosa radi tako da se varijabla red inkrementira kasnije u *if* petlji koja provjerava ispravnost unosa, odnosno jedini slučaj u kojem se ispisuje da je unos netočan je ako je korisnik unio neispravan simbol i varijabla red se nije inkrementirala, te je time jednaka varijabli prosli_red.

```
if prosli_red == red: print("Netočan unos!")
prosli_red=red
```

Kod 5.5 Detekcija neispravnog unosa

Nakon detekcije korisniku se omogućuje unos poteza i taj se podatak sprema u varijablu potez koja se zatim provjerava i obrađuje u nekoliko *if* petlji (Kod 5.6). Prva *if* petlja provjerava je li korisnik unio znak „q“ koji mu omogućuje izlazak iz igre. U slučaju da je unesen taj znak izlazi se iz unutarnje petlje i time trenutna igra završava. Sljedeća *if*, odnosno *elif*, petlja provjerava je li uneseni znak broj i je li taj broj jedan od ključeva u varijabli ploca, drugim riječima jedini znakovi koji zadovoljavaju oba uvjeta su brojevi od jedan do devet. Ako su oba uvjeta zadovoljena ulazi se u još jednu *if* petlju koja je unutar prošle i ona ima zadatak provjere je li na tom polju već obavljen potez. Ako potez nije obavljen na tom se polju tada izvršava program te zadnje *if* petlje.

```
izbor = input()

if izbor == 'q': igra = False
elif str.isdigit(izbor) and int(izbor) in ploca:
    if not ploca[int(izbor)] in {"X", "O"}:
```

Kod 5.6 Provjere unosa korisnika

Dio programa unutar zadnje *if* petlje (Kod 5.7) ima nekoliko funkcija. Prva je upis znaka u varijablu ploča na mjesto unosa korisnika. Prema liniji programskog koda vidi se kako će biti upisna povratna vrijednost funkcije provjeri_red (Kod 5.8). Ta funkcija može vratiti samo jedan od dva *stringa*, a to su „X“ ili „O“ što se provjerava ostatkom dijeljenja varijable red s dva. Drugim riječima ako je vrijednost varijable red neparan broj funkcija vraća „O“, a ako je vrijednost paran broj tada vraća „X“. Rezultat toga svega je da se u varijablu ploča element kojem se pristupa ključem koji je jednak broju kojeg je unio korisnik mijenja u „X“ ili „O“ ovisno o tome koji je igrač bio na redu.

```
ploca[int(izbor)] = provjeri_red(red)

if pobjeda(ploca) == True:
   igra = False
   kraj = True

if red>8: igra=False

comm(izbor,red,ploca)

red += 1
```

Kod 5.7 Programski kod koji se izvodi nakon ispravnog unosa korisnika

Promjenom vrijednosti odabranog elementa varijable ploca dvije *if* petlje provjeravaju je li došlo do kraja igre i ako jest je li igra završila izjednačenim rezultatom ili pobjedom jednog od igrača. Prva petlja koristi funkciju pobjeda koja prolazi kroz sve moguće scenarije koje rezultiraju

pobjedom i vraća *true,* ako je barem jedan od njih ispunjen (Kod 5.9). Ta funkcija provjerava imaju li neke od dijagonala, vertikala ili horizontala tri jednaka znaka. U slučaju da taj uvjet ispunjen varijabla igra postavlja se kao *false* što će značiti da se unutarnja *while* petlja ne izvršava u idućoj iteraciji i postavlja varijablu kraj kao *true* što će poslije biti korišteno kod ispisa informacije o tome je li igra završila pobjedom igrača ili izjednačeno. Druga petlja provjerava vrijednost varijable red. Ako je vrijednost veća od 8, tj. kada dosegne vrijednost devet, to znači da su sva polja ispunjena i igra je završena. Ako nije ispunjen ni jedan uvjet za pobjedu varijabla kraj ostaje *false* i samo se varijabla igra mijenja u *false* što također rezultira neizvođenjem petlje u idućoj iteraciji.

```
def provjeri_red(red):
    if red % 2 == 0: return 'X'
    else: return 'O'
```

Kod 5.8 Funkcija provjeri_red

```
def pobjeda(ploca):
    if (ploca[1] == ploca[5] == ploca[9]) \
    or (ploca[3] == ploca[5] == ploca[7]): return True

    if (ploca[1] == ploca[2] == ploca[3]) \
    or (ploca[4] == ploca[5] == ploca[6]) \
    or (ploca[7] == ploca[8] == ploca[9]): return True

    if (ploca[1] == ploca[4] == ploca[7]) \
    or (ploca[2] == ploca[5] == ploca[8]) \
    or (ploca[3] == ploca[6] == ploca[9]): return True

    else: return False
```

Kod 5.9 Provjera pobjede igrača

Nakon provjere kraja igre poziva se funkcija *comm* koja je zadužena za komunikaciju s klijentom (Kod 5.10). Njoj se kao argumenti šalju varijable izbor, red i ploca iz kojih se izvlače podaci koje je potrebno poslati klijentu. Na početku te funkcije deklarira se varijabla slanje koja će se koristiti u *while* petlji za slanje i pridodaje joj se *boolean* vrijednost *true*. Nakon toga potrebno je provjeriti hoće li zadnji potez rezultirati krajem igre. To se u programu provjerava *if* petljama od kojih prva kao i prije poziva funkciju pobjeda koja provjerava sve moguće scenarije za pobjedu. Osim što se u ovom slučaju rezultat te funkcije sprema u varijablu *end*. U slučaju da je zadnji potez rezultirao pobjedom igrača *end* poprima vrijednost „1“, a u suprotnom poprima vrijednost „0“.

Druga *if* petlja gleda je li popunjena cijela ploča, tj. je li igra završila izjednačeno što također postavlja vrijednost varijable *end* u „1“.

```
def comm(izbor,red,ploca):
     slanje=True

     if pobjeda(ploca) == True: end="1"
     else: end="0"

     if red>7: end="1"
```

Kod 5.10 Postavljanje potrebnih varijabli

Sljedeći je korak ulazak u *while* petlju koja radi tako dugo dok varijabla slanje ima vrijednost *true* otvaranje servera i uspostavljanje veze s klijentom (Kod 5.11). Za rad ovog programa nije nužno stavljati komunikaciju unutar petlje jer se komunicira samo s jednim uređajem. Kada je potrebna komunikacija s više uređaja potrebno je imati dio programa za komunikaciju u petlji kako bi se izvodio isti kod ali s drugim uređajem. U ovome programu je stavljeno u *while* petlju jer funkcionalno ne mijenja ništa i jer je to vrlo česti oblik pisanja. Prvo se stvara *socket* nazvan *ServerSocket*. Zatim mu je potrebno naredbom *bind* pridodati IP adresu i port. To je ista IP adresa i port na koji klijent šalje zahtjev za konekciju. Nakon toga naredbom *listen* se „osluškuje“ mreža za zahtjevima za konekcijom i na kraju se naredbom *accept* prihvaća zahtjev za konekciju i tom se *client socketu* daje ime *client*. Njegova se adresa sprema u varijabli *addr*, ali se za potrebe ovog programa ne koristi.

```
while slanje:

  Port=5000
  ServerSocket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  ServerSocket.bind(("192.168.0.1", Port))
  ServerSocket.listen(2)
  client, addr = ServerSocket.accept()
```

Kod 5.11 Postavljanje servera

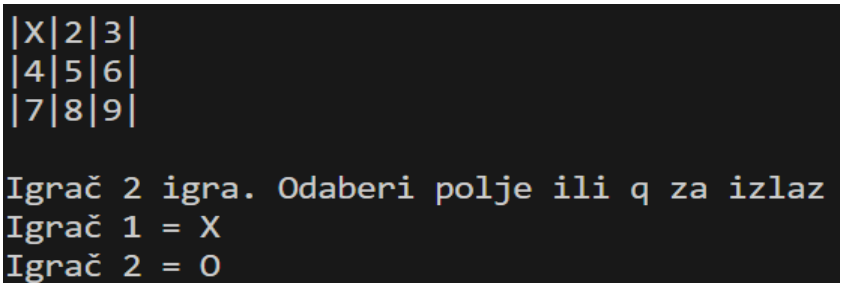
Nakon toga konekcija je uspostavljena i program kreće sa slanjem potrebnih podataka (Kod 5.12). Redoslijed naredbi usklađen je s redoslijedom naredbi sa strane klijenta, tj. klijent je onaj koji prvi šalje poruku i ona se sprema u varijablu *from_client*. Nakon toga klijentu se šalje varijabla izbor, odnosno odigrano polje. Potom klijent šalje poruku koja se ponovo sprema u varijablu *from_client*. Nakon toga klijentu se šalje informacija o tome koji je igrač napravio potez. Ta se

informacija oblikuje kao dodavanje jedan ostatku dijeljenja varijable red sa dva pretvoren u *string*, te će tako klijentu biti poslano „1“ ili „2“ koje on mora samostalno interpretirati. Klijent zatim šalje informaciju koja se ponovno sprema u varijablu *from_client*. Ta se varijabla tako svaki puta prepiše s novom vrijednošću, ali to nije toliko bitno jer se koristi tek nakon zadnje poruke dobivene od klijenta. Zadnji podatak koji se šalje klijentu je informacija o pobjedi, tj. varijabla *end*. Kada klijent primi taj podatak kreće s pomakom potrebne figure dok poslužitelj čeka. Kako korisnik ne bi mislio da je nešto pošlo po krivu ispisuje se tekst „Čekanje robota“. Program potom čeka natrag poslanu poruku koju, kada stigne, sprema u varijablu *from_client*. *If* petlja gleda je li ta varijabla jednaka tekstu „*Zavrseno*“ što mijenja vrijednost varijable slanje u *false* što će spriječiti ponovnu iteraciju petlje. Prije izlaska iz petlje naredbom *close* zatvara se konekcija i oba programa nastavljaju rad.

```
from_client=client.recv(Port).decode()
client.send(izbor.encode())
from_client=client.recv(Port).decode()
client.send(str(red%2+1).encode())
from_client=client.recv(Port).decode()
client.send(end.encode())

print("Čekanje robota")
from_client=client.recv(Port).decode()

if from_client=="Zavrseno":slanje=False
ServerSocket.close()
```

Kod 5.12 Razmjena komunikacija sa klijentom

Time završava izvođenje programa funkcije *comm* i program se vraća u unutarnju *if* petlju. Zadnja linija koda prije završetka iteracije unutarnje *if*, a time i unutarnje *while*, petlje je inkrementacija varijable red kako bi se, ako igra nije završila, promijenila varijabla red što će omogućiti promjenu znaka koji se upisuje i ispravnost rada dijela programa koji provjerava ispravnost unosa. Ukoliko igra nije završila zadnjim potezom program ponovno prolazi kroz unutarnju *while* petlju i cijeli se proces ponavlja. Za razliku od prve iteracije sljedeće crtanje ploče za igranje u konzoli ima umjesto brojeva „X“ ili „O“ na mjestu koje je odabrao igrač (Slika 5,2).



Slika 5.2 Primjer druge iteracije gdje je prvi igrač izabrao polje broj jedan

Ukoliko je igra završila zadnjim potezom izlazi se iz glavne petlje. Prvo se izlaskom iz petlje briše tekst konzole i crta završno stanje igre, a dio teksta koji daje korisniku informacije određen je dvjema *if* petljama, dok ostatak prikazanog teksta ne ovisi o tome kako je završila igra (Kod 5.13).

```
os.system ('cls' if os.name == 'nt' else 'clear')
crtanje(ploca)

if kraj == True:
    if provjeri_red(red) == 'X': print("Pobjedio je igrač 2")
    else: print("Pobjedio je igrač 1")

if kraj == False: print("Izjednačeno")

print("Igrač 1 = X")
print("Igrač 2 = O")
print("Igra je završena")

print("Unesi p za ponovnu igru ili bilo koji drugi znak za izlaz")
izbor=input()
if izbor != "p": break
```

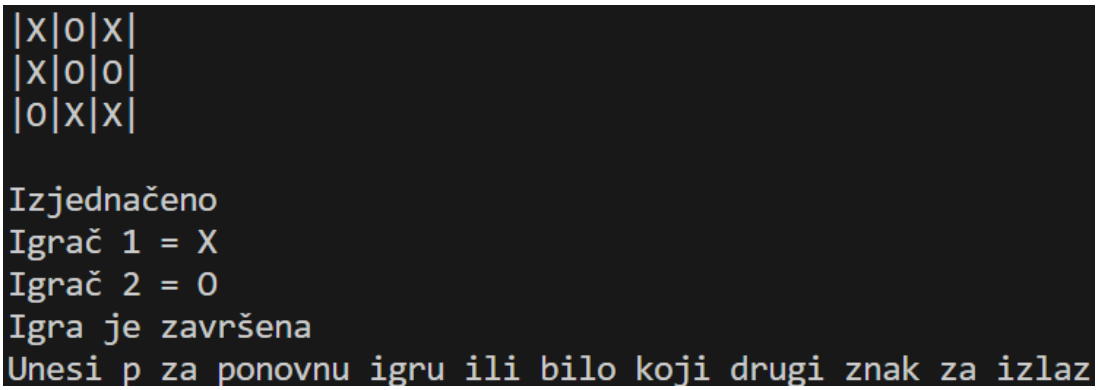
Kod 5.13 Ispis rezultata i upute za ponovno pokretanje igre

Varijabla koja odlučuje koji se tekst ispisuje je varijabla kraj. Ako je igra završila pobjedom igrača (Slika 5.3) tada se izvodi program prve *if* petlje gdje se pozivom funkcije provjeri_red provjerava koji je igrač pobjednički. Zbog toga što se prije izlaska iz petlje inkrementira varijabla red potrebno je drugačije interpretirati podatke koje vraća funkcija provjer_red. Kroz cijeli se program igrač „križić“ tretira kao igrač 1, a igrač „kružić“ kao igrač 2. Zbog promjene varijable red, ako funkcija provjeri_red vrati podatak „X“ potrebno je ispisati da je pobjednik igrač 2, a ne igrač 1. Također vrijedi i suprotan slučaj.

```
|X|O|X|
|X|O|6|
|7|O|9|

Pobjedio je igrač 2
Igrač 1 = X
Igrač 2 = O
Igra je završena
Unesi p za ponovnu igru ili bilo koji drugi znak za izlaz
```

Slika 5.3 Ispis kod pobjede igrača

U slučaju da varijabla kraj kod izlaska iz petlje ima vrijednost *false* ne poziva se nikakva funkcija već se samo ispisuje kako je igra završila izjednačeno (Slika 5.4). Osim tih mogućih informacija ispisuju se ponovo informacije koji igrač ima koji znak. Na samome kraju se također ispisuje tekst koji daje uputu korisniku da može unijeti znak p za ponovnu igru ili bilo koji drugi znak za završetak igre. Taj unos se sprema u varijablu izbor jer se za potrebe igre više ne koristi, te se odmah nakon unosa provjerava varijabla izbor. U slučaju da je unesen bilo koji simbol osim „p“ izvodi se naredba break kojom se izlazi iz vanjske *while* petlje programa i program završava. U slučaju da je korisnik unio simbol „p“ ponovno se izvodi vanjska *while* petlja u kojoj se postavljaju potrebne varijable na potrebne vrijednosti i proces kreće od početka.



Slika 5.4 Ispis kod neriješene igre

6. Zaključak

Kombinacijom RAPID i Python programska jezika, ili u drugim projektima nekih drugih programskih jezika, povećava se efikasnost cijelog projekta odnosno sustava. Iskorištavanjem prednosti RAPID programskog koda, koji je stvoren za kontrolu industrijskih robota, i Python programskog koda, koji svojom fleksibilnošću može lako obaviti širok spektar problema, dolazi do povećanja mogućnosti izgradnje kompleksnijih sustava upravljanja. Bez komunikacije *socketima* mogućnost sinergije dva različita programska jezika bila bi znatno teža. Takvom komunikacijom omogućuje se koordinirana i laka razmjena podataka potrebnih za rad projekta dok ostalo vrijeme programi izvršavaju zasebne radne zadatke. Ovaj se projekt uz neke poteškoće možda i mogao izvesti sa samo jednim programskim jezikom, te služi kao dokaz koncepta. Kod već malo kompliciranijih projekata korištenje više programskih jezika postaje nužno. Takav pristup projektiranju većih sustava može povećati njihovu učinkovitost i omogućiti brzu optimizaciju i fleksibilnost koja ne bi postojala kada bi svaki element sustava radio zasebno.

7. Literatura

[1] https://library.e.abb.com,

[2]https://docs.oracle.com/

[3]https://www.digitalocean.com/

[4] https://www.scaler.com/

[5] https://developercenter.robotstudio.com/

Popis slika

Slika 2.1 Pojednostavljeni dijagram toka procesa 3
Slika 2.2 Robotska ruka i krajnji efektor u razvojnom okruženju RobotStudio 4
Slika 2.3 Figura za igranje 5
Slika 2.4 Ploča za figure 6
Slika 2.5 Podmetači za fiksiranje s prikazanim nevidljivim bridovima 6
Slika 2.6 Ploča za igranje 7
Slika 2.7 Postavljeni elementi za igru u programskom okruženju RobotStudio 7
Slika 2.8 Približeni prikaz pričvršćivanja ploče na stol 8
Slika 3.1 Dijagram toka socket programiranja [4] 9
Slika 4.1 Okolina s prikazom radnog obujma 11
Slika 4.2 Shematski prikaz workobjecta i meta 12
Slika 5.1 Ispis u terminalu kod pokretanja programa 28
Slika 5.2 Primjer druge iteracije gdje je prvi igrač izabrao polje broj jedan 32
Slika 5.3 Ispis kod pobjede igrača 33
Slika 5.4 Ispis kod neriješene igre 34

Popis primjera kodova

Kod 4.1 Deklaracije workobjecta i robtargeta 13
Kod 4.2 Deklaracija varijabli za komunikaciju 14
Kod 4.3 Deklaracija pomoćnih varijabli 15
Kod 4.4 Main petlja 15
Kod 4.5 Razmjena podataka 16
Kod 4.6 Obrada grešaka 17
Kod 4.7 Određivanje točke uzimanja križić figure 17
Kod 4.8 Određivanje točke za uzimanja kružić figure 18
Kod 4.9 Određivanje točke spuštanja figure za igranje 19
Kod 4.10 Uzimanje križić figure s ploče za figure 19
Kod 4.11 Rutina za otvaranje čeljusti prihvatnice 20
Kod 4.12 Rutina za zatvaranje čeljusti prihvatnice 20
Kod 4.13 Uzimanje kružić figure s ploče za figure 20
Kod 4.14 Postavljanje figure na ploču za igranje 21
Kod 4.15 Zapis odigranog poteza 22
Kod 4.16 For petlja rutine 22
Kod 4.17 Biranje polja s kojeg se uzima figura 23
Kod 4.18 Biranje praznog mjesta za pospremanje na ploči za križić figure 23
Kod 4.19 Biranje praznog mjesta za pospremanje na ploči za kružić figure 23
Kod 4.20 Uzimanje figure koja se posprema 24
Kod 4.21 Pospremanje križić figura 24
Kod 4.22 Pospremanje kružić figura 25
Kod 4.23 Brisanje zapisa poteza 25
Kod 5.1 Uvezene funkcije 26
Kod 5.2 Deklaracija varijabli 27
Kod 5.3 Crtanje ploče u konzoli i ispis informacija 27
Kod 5.4 Funkcija za crtanje ploče u konzolu 28
Kod 5.5 Detekcija neispravnog unosa 28
Kod 5.6 Provjere unosa korisnika 29
Kod 5.7 Programski kod koji se izvodi nakon ispravnog unosa korisnika 29
Kod 5.8 Funkcija provjeri_red 30
Kod 5.9 Provjera pobjede igrača 30
Kod 5.10 Postavljanje potrebnih varijabli 31

Kod 5.11 Postavljanje servera 31
Kod 5.12 Razmjena komunikacija sa klijentom 32
Kod 5.13 Ispis rezultata i upute za ponovno pokretanje igre 33

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Kevin Gržan _____ *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog/specijalističkog (*obrisati nepotrebno*) rada pod naslovom Povezivanje AC računala i računalne kontrolera pomoću programskog jezika Python *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.